Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Interaktivní nástroj pro podporu výuky úvodu do matematické logiky

An interactive tool for teaching introduction to Mathematical logic

2020 **Martin Kopelec**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Martin Kopelec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Interaktivní nástroj pro podporu výuky úvodu do matematické logiky
An Interactive Tool for Teaching Introduction to Mathematical Logic.

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je text a interaktivní sw nástroj, sloužící jako didaktická pomůcka k problematice úvodu do matematické logiky. Doprovodná aplikace bude implementována pro mobilní zařízení s operačním systémem Android 8 nebo vyšší.

Práce bude obsahovat:
1. Teorii základů výrokové logiky (VL).
2. Teorii sémantiky VL a sémantických důkazů.
3. Teorii syntaktických postupů pro ekvivalentní transformace, taktéž do KNF a DNF.
4. Implementace a otestování interaktivnho softwareu, který bude sloužit jako didaktická pomůcka k teorii úvodu do logiky pro studenty bakalářského stupně studia. Výsledná aplikace bude pro mobilní zařízení s OS Android 8 nebo vyšší.

Seznam doporučené odborné literatury:

[1] Duží M.: *Logika pro informatiky a příbuzné obory*. VŠB-Technická universita Ostrava, 2012, ISBN 978-80-248-2662-2
[2] ŠVEJDAR, Vítězslav. *Logika, neúplnost, složitost a nutnost*. Praha: Academia, 2002. ISBN 978-802-0010-056.
[3] LACKO, Ľuboslav. *Mistrovství - Android*. Brno: Computer Press, 2017. Mistrovství. ISBN 978-802-5148-754.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Marek Menšík, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *14. května 2020*

.....
podpis studenta

Poděkování

Velice rád bych poděkoval vedoucímu práce Mgr. Marku Menšíkovi, Ph.D. za odbornou pomoc, přínosné konzultace, velkou trpělivost a také konstruktivní kritiku při vytváření této bakalářské práce. Dále bych chtěl také poděkovat všem přátelům a rodině za velikou podporu po celou dobu studia.

Abstrakt

Cílem mé bakalářské práce je vytvořit text a interaktivní softwarový nástroj, který bude v budoucnu sloužit jako didaktická pomůcka pro výuku základů výrokové logiky. Stěžejní částí bakalářské práce je interaktivní software, který by měl pomoct jednodušeji objasnit a procvičit danou problematiku za pomocí jednoduchého ovládání v podobě mobilní aplikace na operačním systému Android 8 nebo vyšším. Studenti by se poté mohli snáze dostat do kontaktu s danou problematikou a procvičovat si řešení, což by mohlo vést k efektivnějšímu způsobu edukace studentů.

Interaktivní software vychází z všeobecných základů oboru výrokové logiky, a jeho úkolem je řešit jednotlivé syntaktické úpravy se zadaným vstupem, jenž by měla být formule výrokové logiky. Pro názornost můžeme uvést: převedení formule výrokové logiky do KNF/DNF/ÚKNF/ÚDNF, generování pravdivostní tabulky dané formule, či ověření správnosti ekvivalentních úprav nad formulemi.

Text bakalářské práce se postupně zabývá teorií základů výrokové logiky, dále teorií sémantiky výrokové logiky a sémantických důkazů a v neposlední řadě také teorií syntaktických postupů pro ekvivalentní transformace a poukazuje na využití daných teorií a způsobů ve výše zmíněném interaktivním softwaru. Na konec je pozornost věnována funkcím a fungování výsledného softwaru.

Klíčová slova

Výroková logika, logika, interaktivní nástroj, software, didaktická pomůcka, Java, ekvivalentní transformace

Abstract

The purpose of my bachelor thesis is to create text and interactive software which will be an didactit tool for a teaching of basics of propositional logic in the future. Major part of this bachelor thesis is to create an interactive software, that can help to better understanding as a mobile application for Android operating system with version 8.0 or higher. With this interactive tool students could practice many of functions and solutions in propositional logic, which could lead to a better and more effective way of educating students.

Interactive software is based on the general basics of propositional logic and its methods to solving individual syntactic modifications with the given input which should be any formula of propositional logic. For example we can mention: transforming given formula to the DNF/CNF/CDNF/CCNF, generation of truth table for a given formula or verification of the correctness of equivalent transformations.

The text of this bachelor thesis gradually deals with the theory of basics of propositional logic, the theory of semantics of propositional logic and semantic proofs and last but not least the theory of syntactic procedures for equivalent transformations. The text also points to the use of these theories and methods in interactive software and describes the mechanics of the software and its functions.

Key words

Propositional logic, logic, interactive tool, software, didactic tool, Java, equivalent transformations

Seznam použitých symbolů a zkratek

Zkratka/Symbol	Význam
DNF	Disjunktivní normální forma
ED	Elementární disjunkce
EK	Elementární konjunkce
KNF	Konjunktivní normální forma
UDNF	Úplná disjunktivní normální forma
UED	Úplná elementární disjunkce
UEK	Úplná elementární konjunkce
UKNF	Úplná konjunktivní normální forma
VL	Výroková logika
¬	Symbol negace
!	Alternativní symbol pro negaci
~	Alternativní symbol pro negaci
∧	Symbol konjunkce
&	Alternativní symbol pro konjunkci
∨	Symbol disjunkce
V	Alternativní symbol pro disjunkci
⊃	Symbol implikace
→	Alternativní symbol pro implikaci
⇒	Alternativní symbol pro implikaci
≡	Symbol ekvivalence
↔	Alternativní symbol pro ekvivalenci
⇔	Alternativní symbol pro ekvivalenci
|=	Symbol pro tautologii
&&	Symbol pro logickou operaci AND
||	Symbol pro logickou operaci OR

Obsah

Seznam tabulek

Seznam obrázků

Úvod

Výroková logika je nedílnou součástí studijních osnov na mnoha vysokých školách. Tato práce se zabývá stručným úvodem do základů výrokové logiky, její sémantiky a také teorií ekvivalentních úprav. Cílem této práce je implementovat interaktivní softwarový nástroj, který by mohl pomoci ve výuce výrokové logiky jako takové, a pomoct tak jejímu lepšímu porozumění. Pro implementaci takového interaktivního softwaru je však nejprve nutné říct si o tom, co to výroková logika je a o její základní teorii. Výroková logika má svou syntaxi a sémantiku, a tak bude pozornost věnována taktéž těmto pojmům. Pro představení výrokové logiky a také základních pojmů bude věnována první polovina této práce.

Pro operace, které budou následně implementovány v interaktivním softwaru, je důležité se opřít o teorii a teoretické postupy, které se ve výrokové logice uplatňují a používají, a kterým bude v této bakalářské práci věnována hlavní pozornost. Po představení tohoto teoretického základu pak bude následovat část věnovaná samotné implementaci softwaru a následnému možnému využití tohoto softwaru ve výuce. Této části jsou věnovány kapitoly 3. a 4., které jsou rovněž závěrem této práce, a tak i představením samotného softwaru a jeho otestování v praxi.

V praktické části aneb v implementaci samotného programu jsou realizovány základní funkce pro práci s formulemi VL, jakými jsou například generace pravdivostní tabulky, generace normální forem, výběr a provedení daných ekvivalentních transformací nad zadanými formulemi či kontrola správnosti provedení daných ekvivalentních úprav nad formulemi výrokové logiky. To vše s grafickým rozhraním a optimalizací pro dnešní mobilní zařízení.

1 Teorie základů výrokové logiky

1.1 Co je to výroková logika a k čemu slouží

Výroková logika (dále označována zkratkou **VL**) bývá často charakterizována jako "*logika zkoumající logické vztahy mezi výroky*" [1], jak již popsal Jiří Raclavský ve své knize *Úvod do logiky: klasická výroková logika*. Je tedy na místě popsat, co lze považovat za výrok. Za **výrok** lze považovat větu či tvrzení, o které můžeme prohlásit, zda je pravdivá či nepravdivá. Na takovéto tvrzení můžeme tedy odpovědět pouze dvěma možnostmi: ano/ne, pravda/nepravda, true/false či 1/0. Takový fakt nabízí myšlenku, že takovéto logické rozhodování jde ruku v ruce s **Booleovou logikou**. Ve skutečnosti tomu tak opravdu je. VL je v dnešním světě všude kolem nás. Je totiž nedílnou součástí v logických obvodech, které jsou široce využívány v elektrotechnice. K tomuto tématu však ale navážeme později. Výrok je tedy věta, avšak každá věta nemusí být výrok. Jak jsem již zmínil, na výrok lze odpovědět, jestli je pravdivý, či nepravdivý. Avšak například věta: "Současný český král hraje fotbal." nám neumožňuje určit, zda je tato věta pravdivá, či nepravdivá. Z těchto poznatků vyplývá, že klasická výroková logika – pracující se dvěma pravdivostními hodnotami ctí **princip dvojhodnotovosti**., který říká: *"Každý výrok je pravdivý, nebo nepravdivý."* [1]

Výroky, jako takové rozdělujeme na výroky **jednoduché** a **složené**. Jednoduchý výrok je takové tvrzení, jehož části již nevyjadřují žádný další výrok. Jednoduché výroky jinak nazýváme také jako **elementární** výroky. **Složený** výrok je naopak takové tvrzení, jehož části jsou jednotlivými výroky. [2]

VL se však nesoustředí a nezabývá strukturou elementárních výroků, jelikož můžeme tyto výroky chápat pouze jako pravdivé či nepravdivé tvrzení. VL se tedy soustředí a zabývá **strukturou složených výroků**. Ten totiž vzniká skládáním a spojováním elementárních výroků. Tyto jednotlivé elementární výroky však musí být něčím spojovány, právě k tomu nám slouží **logické spojky**. Nyní se může nabízet otázka: Jak je to s pravdivostní hodnotou u složených tvrzení? Pravdivostní hodnota je v tomto případě závislá na pravdivostních hodnotách jednotlivých pod-výroků a jejich spojek, které jednotlivé tvrzení spojují, jak již bylo zmíněno výše. Proto VL uplatňuje tzv. **Princip kompozicionality**, který říká: *"Pravdivostní hodnota výroku je jednoznačně určena pravdivostními hodnotami jeho složek, tj. pravdivostními hodnotami dílčích výroků a sémantikou spojek, jež tyto dílčí výroky spojují."* [1]

Jak je již dobře známo, každý jazyk by měl být definován abecedou a gramatikou, kde abeceda určuje, jaký typ symbolu můžeme v daném jazyce použít a gramatika nám definuje daná pravidla pro tvorbu dobře utvořeného výrazu odpovídající danému jazyku [2]. Abeceda jazyka výrokové logiky musí obsahovat symboly, které budou zastupovat jednotlivé jednoduché výroky (proměnné), symboly pro logické spojky a popřípadě pomocné symboly. Výjimkou není ani jazyk VL.

1.2 Jazyk výrokové logiky

Jak jsme se již dozvěděli v minulé podkapitole, každý jazyk by měl mít definovanou **abecedu** a také **gramatiku**. **Abeceda** jazyka výrokové logiky nám jasně definuje, jaké symboly spadají do množiny používaných symbolů VL, které jsou následující:

- **Symboly pro výrokové proměnné** – a, b, c, d, e…p, q, r… (je zvykem používat písmena abecedy, případně s indexy – velká, či malá)
- **Symboly pro logické spojky** (též funktory)
    - ¬ negace
    - ∧ konjunkce
    - ∨ disjunkce
    - ⊃ implikace
    - ≡ ekvivalence
- **Pomocné symboly** (závorky)

Gramatika nám určuje způsob, jakým definujeme formule VL. Tím pádem může sloužit také jako návod, jak **správně utvořit formuli VL**. [3] Když jsem tedy zamýšlel tvorbu interaktivního programu, který bude pracovat právě se správně utvořenými formulemi, bylo mi hned jasné, že základem pro správný vstup a kontrolu vstupu bude právě **gramatika** jazyka. Gramatika nám tedy rekurzivně definuje nekonečnou množinu formulí:

- 1. Báze definice – Výrokové symboly jsou formule.
- 2. Indukce – Jsou-li A, B formule, jsou formulemi i tyto výrazy: (¬ A), (A ∧ B), (A ∨ B), (A ⊃ B), (A ≡ B)
- 3. Generalizace – Není jiných formulí VL, než podle bodu 1. a 2. [3] [2]

K těmto bodům bychom si také měli uvést, že formule, které jsou vytvořeny podle bodu **1.** jsou formulemi **elementárními** (jednoduchými) a formule utvořené dle bodu **2.** jsou formulemi **složenými**. A tedy pro shrnutí, jak uvádí prof. Marie Duží ve své publikaci *Logika pro informatiky (a příbuzné obory)*: *"Jazyk výrokové logiky je množina všech formulí výrokové logiky."* [2] Když jsme však pomocí abecedy jazyka VL uváděli symboly výrokových spojek, tak je na místě uvést, že tyto spojky mohou mít více variant značení. Zde je tabulka s alternativními symboly:

Tabulka 1.1: *Alternativní symboly výrokových spojek*

Symboly spojek	¬	∧	∨	⊃	≡
Alternativní symboly	~, !	&	V	→, ⇒	↔, ⇔

1.2.1 Rozbor výrokových spojek

Když si vzpomeneme na to, že výrok je tvrzení, o kterém se dá prohlásit, zda je pravdivé či nepravdivé, a že složené výroky vznikají spojením jednotlivých výroků pomocí výrokových spojek, tak můžeme poukázat na to, že výrokové spojky jsou jakousi logickou variantou spojek v přirozeném jazyce.

- Spojka **negace** "¬" je vyjadřována jako "**ne**", či "**není pravda, že**". Je to **unární** operace což značí, že má pouze jeden operand [4]. Příkladem z přirozeného jazyka může být věta: *"Není pravda, že jsem byl běhat."*

Tabulka 1.2: *Pravdivostní tabulka negace*

p	**¬p**
0	1
1	0

- Spojka **konjunkce** "∧" je vyjadřována jako "a". Je to **binární** operace, tudíž má dva operandy a je také komutativní což znamená, že u této spojky nezáleží na pořadí jejich operandů. Příkladem z přirozeného jazyka může být věta: *"Je mladý a je pilot."*

Tabulka 1.3: *Pravdivostní tabulka konjunkce*

p	**q**	**p ∧ q**
0	0	0
0	1	0
1	0	0
1	1	1

- Spojka **disjunkce** "∨" je vyjadřována jako "nebo". Jde o binární a komutativní spojku. Příkladem z přirozeného jazyka může být věta: *"Auto je pojízdné nebo není pojízdné."*

Tabulka 1.4: *Pravdivostní tabulka disjunkce*

p	**q**	**p ∨ q**
0	0	0
0	1	1
1	0	1
1	1	1

- Spojka **implikace** "⊃" je vyjadřována jako "jestliže, pak". Je to **binární** a **nekomutativní** spojka. První člen implikace se nazývá antecedent a druhý člen se nazývá konsekvent. Příkladem z přirozeného jazyka může být věta: *"Jestliže jsem král, pak můžu rozkazovat."*

Tabulka 1.5: *Pravdivostní tabulka implikace*

p	**q**	**p ⊃ q**
0	0	1
0	1	1
1	0	0
1	1	1

- Spojka **ekvivalence** "≡" je vyjadřována jako "tehdy a jen tehdy" či "právě tehdy, když". Je to binární a komutativní spojka. Příkladem z přirozeného jazyka může být věta: *"Jsem spokojený právě tehdy, když sportuji."* [2]

Tabulka 1.6: *Pravdivostní tabulka ekvivalence*

p	**q**	**p ≡ q**
0	0	1
0	1	0
1	0	0
1	1	1

2 Teorie sémantiky VL a sémantických důkazů

2.1 Sémantika a sémantické pojmy

Sémantika VL jako taková má za úkol přiřazovat hodnotu či význam jednotlivým tvrzením – formulím. Pro nás to zkrátka znamená přiřazení pravdivostní hodnoty, která, jak už víme, nabývá dvou hodnot – pravda/nepravda [1]. Pokud však chceme přiřadit hodnotu nějaké formule musíme nejprve specifikovat, za jakých podmínek – v našem případě za jakého ohodnocení k přiřazení hodnoty dané formule došlo.

Pravdivostní ohodnocení neboli také valuace je funkce, která přiřazuje každému výrokovému symbolu pravdivostní hodnotu a to právě 0 nebo 1. Pomocí symbolů s přiřazenou hodnotou poté můžeme vyhodnotit celkovou množinu – pravda/nepravda. [2]

Při ohodnocování formulí se nám, jak již víme, nabízejí různá ohodnocení (valuace). Dané ohodnocení, při kterém nabývá určitá formule hodnoty 1/true ve VL nazýváme modelem. Tudíž každé jednotlivé ohodnocení, při kterém nabývá formule hodnoty 1 je modelem [2]. Když jsme se již dozvěděli, co znamená pojem model je na místě uvést další důležité pojmy, jako jsou splnitelnost, tautologičnost a kontradikčnost. Všechny tyto pojmy jsou úzce spjaty s ohodnocením dané formule. Lépe řečeno jsou přímo odvozovány od ohodnocení [1].

Splnitelnost formule – formule je splnitelná právě tehdy, když je alespoň jedno její ohodnocení modelem. Jinými slovy existuje alespoň jedno ohodnocení, při kterém je konkrétní formule pravdivá. [5]

Tautologie – formule je tautologií právě tehdy, když jsou všechna její ohodnocení modelem. Jinými slovy neexistuje ani jedno ohodnocení, při kterém by byla formule nepravdivá. Pro označení tautologie pro formuli A používáme "|=A" [1].

Kontradikce – formule je kontradikcí právě tehdy, když neexistuje ani jeden model formule. Jinými slovy pro všechna ohodnocení je formule nepravdivá.

Je také na místě uvést definici výrokově logického vyplývání, která říká: *"Formule A výrokově logicky vyplývá z množiny formulí M, značíme M |= A, jestliže A je pravdivá v každém modelu množiny M."* [2].

V následující tabulce můžeme vidět příklady tří formulí a jejich ohodnocení. Formule "p ∨ ¬p", které náleží druhý sloupec je tautologie – všechna ohodnocení jsou pravdivá. Formule "p ∨ p", které náleží třetí sloupec je splnitelná – má alespoň jeden model. Formule "p ∧ ¬p" v posledním sloupci je kontradikcí – formule nemá ani jeden model.

Tabulka 1.7: *Ukázka vlastností daných formulí*

p	**p ∨ ¬p**	**p ∨ p**	**p ∧ ¬p**
0	1	0	0
1	1	1	0

Dalším důležitým pojmem je **dualita**. Duální formuli k formuli A dostaneme tím způsobem, že si nejprve vytvoříme pravdivostní tabulku formule A. Když všechny pravdivostní hodnoty této tabulky zaměníme za opačnou hodnotu, dostaneme pravdivostní tabulku formule $\boldsymbol{A^D}$, která je **duální** k naší formuli A [9]. Dualitu si názorně ukážeme na následujícím příkladu těchto dvou tabulek.

Tabulka 1.8: *Pravdivostní tabulka formule A*

p	**q**	**p ∧ q**
0	0	0
0	1	0
1	0	0
1	1	1

Tabulka 1.9: *Pravdivostní tabulka formule $\boldsymbol{A^D}$ duální k formuli A*

p	**q**	$(\mathbf{p} \wedge \mathbf{q})^{\boldsymbol{D}}$
1	1	1
1	0	1
0	1	1
0	0	0

Když nyní porovnáme tyto dvě tabulky, zjistíme, že se tabulky liší pouze v pořadí jejich řádků. Operace konjunkce a disjunkce tedy nazýváme duálními operacemi. Definice 1 (duální formule): *"Formule A^D, jejíž interpretační tabulka byla vytvořena z interpretační tabulky formule A vzájemnou záměnou jejich pravdivostních hodnot, je formulí duální k formuli A."* [9].

Na konec této podkapitoly si uvedeme, co znamená pravidlo substituce ve VL. Definice 2 (pravidlo substituce): *"Nahradíme-li ve VL – tautologii každý výskyt určité proměnné určitou jednou a toutéž formulí, zůstane VL – tautologií."* [1]. Což pro nás jednoduše znamená, že například při tautologii "p ⊃ p" můžeme za p dosadit neboli substituovat například "q ∨ r". Výsledky těchto substitucí budou tudíž rovněž tautologií [1].

2.2 Nejpoužívanější sémantické metody výrokové logiky

2.2.1 Tabulková metoda

Pravdivostní vyhodnocení formule bývá nejčastěji interpretováno pomocí tzv. tabulkové metody neboli také **pravdivostní tabulky**, která slouží ke zjištění pravdivostní hodnoty pro každou možnou valuaci dané formule [6].

Zobrazuje nám **jednotlivé ohodnocení (valuace) formule**. Velikost této tabulky je závislá na **počtu proměnných** – pro nás tedy na počtu jednotlivých výrokových proměnných. Pokud budeme mít právě **n proměnných**, tak naše pravdivostní tabulka bude mít právě $\boldsymbol{n^2}$ řádků. Tento počet nám však také udává, kolik možných kombinací ohodnocení může při dané formuli nastat.

2.2.1.1 ***Implementace pravdivostní tabulky v softwaru***

Pravdivostní tabulka patří k základním prvkům v práci s výrokovou logikou a má široké využití. Proto jsem se ji také rozhodl implementovat v interaktivním softwaru. Hlavním stavebním prvkem pro pravdivostní tabulku je právě počet užitých výrokových proměnných v dané formuli. Od tohoto faktu jsem se také odrazil a nejprve jsem v programu zaznamenal, kolik rozdílných proměnných je v dané formuli. Totožné znaky ve větší četnosti nebyly podstatné – důležité bylo, kolik rozdílných proměnných bylo použito. Toto důležité číslo jsem poté využil pro tvorbu daného počtu jednotlivých logických ohodnocení. Následně již stačilo použít jednoduchý algoritmus, který nám generuje ohodnocení pro daný počet proměnných – jedinečné kombinace 0 a 1. Ve fázi, kdy byla vyhotovena množina daných ohodnocení a následně každá jedinečná kombinace nahradila jim přiřazené proměnné v původní formuli, bylo potřeba danou formuli vyhodnotit – určit, zda je celková formule v daném ohodnocení pravdivá, či nepravdivá. K tomuto úkonu je však třeba vytvořit vlastní vyhodnocovací algoritmus, nebo se nabízí mnohem jednodušší řešení, a tím je využití datového typu boolean. Jelikož boolean jako literál nabývá jen a pouze dvou hodnot – true a false, ale také dokáže pracovat s výše zmíněnými logickými operacemi jako je konjunkce – pro boolean je to operace: "&&" (AND), a také disjunkce – pro boolean operace: "||" (OR) [8]. Operace jako implikace či ekvivalence datový typ boolean přímo nezvládá, a tak byla na místě nutná ekvivalentní úprava do podoby reprezentované pouze logickými operacemi "AND" a "OR".

Boolean tedy dokáže akceptovat jakoukoli formuli, ve které jsou na místo proměnných dosazeny hodnoty true/false, a ve které jsou jako logické operace použity pouze konjunkce či disjunkce, pro nás logické operace – AND/OR.

Pokud tedy přiřadíme do deklarované boolean proměnné formuli v podobě např. "(false && true) || false", tak nám je boolean schopen automaticky vyhodnotit celou formuli, a to nehledě na počet závorek či logických proměnných.

Využil jsem tedy této možnosti. Po nahrazení výrokových proměnných hodnotami false/true a po nahrazení značek pro konjunkce a disjunkce značkami pro logické operace AND a OR následoval už jen převod z datového typu String, kde byla dosavadní formule uložena do datového typu boolean a následné vyhodnocení.

Funkce využívající generaci pravdivostní tabulky tedy funguje tak, že se uživateli vygeneruje a zobrazí pravdivostní tabulka pro zadanou formuli, a následně se program dotazuje, zdali je daná formule splnitelná, tautologií, kontradikcí a jaký má počet modelů. Uživatel tak díky vygenerované pravdivostní tabulce může tyto otázky zodpovědět a program mu jeho výsledky zkontroluje.

Tabulka 1.10: *Příklad pravdivostní tabulky pro formuli: "((p ⊃ q) ∨ r)"*

p	**q**	**r**	**((p ⊃ q) ∨ r)**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Je nutno podotknout, že VL označujeme za **rozhodnutelnou**, protože pravdivostní tabulka je vždy konečná a můžeme tak vždy rozhodnout z hlediska splnitelnosti [7].

Jak jsme se již dozvěděli, tak již zmíněné vlastnosti formule jako **splnitelnost, tautologičnost** a **kontradikčnost** formule jsou jedny z nejdůležitějších a jedny z nejčastěji používaných, co se týče práce okolo vyhodnocování formulí. Proto jsem se je také rozhodl implementovat do svého softwaru.

2.2.1.2 Implementace detekce splnitelnosti, tautologičnosti a kontradikčnosti v softwaru

Implementaci těchto vlastností v softwaru můžeme nalézt při generaci pravdivostní tabulky. Bylo potřeba projít pole, které obsahuje výsledky ohodnocení a detekovat, jestli se v poli vyskytuje alespoň jedno pravdivé ohodnocení pro splnitelnost, dále jestli jsou všechna ohodnocení pravdivá pro tautologii či jestli jsou naopak všechna ohodnocení nepravdivá pro kontradikci. Uživatel je potom na tyto vlastnosti dotazován při generaci příslušné pravdivostní tabulky.

Jelikož tyto vlastnosti můžeme uvést u každé správně utvořené formule, software nám dokáže vždy říct, jestli jsou tyto vlastnosti pro danou formuli pravdivé, či nikoliv.

2.2.2 Metoda protipříkladu – sporem

Víme, že tabulková metoda není jedinou sémantickou metodou, kterou můžeme použít k ověřování logického vyplývání či u úsudku splnitelnosti, kontradikčnosti nebo tautologičnosti [2].

Tabulková metoda má totiž jednu velkou nevýhodu, a to tu, že nabývá "velkých rozměrů". To si můžeme jednoduše představit při libovolné ne zrovna složité formuli, která však bude obsahovat 6 proměnných. Jak jsme se již dozvěděli – velikost pravdivostní tabulky – konkrétně počet jejich řádků se odvíjí od počtu proměnných.

V tomto případě bychom totiž dostali $\mathbf{2^6}$ = **64** řádků pravdivostní tabulky. Takhle velkou tabulku není jednoduché zobrazit, natož se v ní rychle orientovat. Přece jen hledání například jedné nepravdivé valuace v takové tabulce může být velice komplikované a ve výsledku také velmi nepřesné. Právě proto používáme také jiné efektivnější metody, jako je právě zmiňovaná **metoda protipříkladu**.

2.2.2.1 Využití metody protipříkladu k ověření platnosti úsudku

Co se týče využití metody protipříkladu k ověření platnosti daného úsudku, tak **postup** je takový, že ohodnotíme závěr daného úsudku jako **nepravdivý**, přičemž všechny jeho **premisy** jsou **pravdivé** a snažíme se nalézt právě takovou valuaci – všechny premisy pravdivé a závěr nepravdivý [1]. Pokud se nám podaří nalézt takovou valuaci, úsudek je **neplatný**. Pokud právě naopak takovou valuaci nejsme schopni nalézt, dokazuje to, že úsudek je **platný**. V této metodě se tedy pokoušíme nalézt spor.

2.2.2.2 Využití metody protipříkladu k dokazování tautologií

Hlavním využívaným principem v této metodě je tento **zákon**: *"|= A právě když ¬A |="* [1]. Ten nám říká, že pokud je formule A **tautologií**, tak **negace** této formule bude právě **kontradikcí**. Zkrátka, když jsou všechna ohodnocení formule A **pravdivá**, tak po jejich **negaci** musí být všechna logicky **nepravdivá**. Pokud zvážíme tento fakt, nabídne se nám dosti jednoduchá možnost pro dokazování tautologie, kterou si názorně ukážeme na následujícím příkladu:

$$\neg p \supset (q \vee p)$$

Rozhodující operací v této formuli je **implikace**, jelikož naše formule je ve tvaru A ⊃ B. Jak již víme, tak našim cílem je najít **valuaci**, při které je tato formule **nepravdivá**. Také víme, kdy je nepravdivá implikace, a to je v případě, kdy je první člen (antecedent) pravdivý a druhý člen (konsekvent) nepravdivý - "1 ⊃ 0". Pokud tedy dokážeme najít takovou valuaci v naší formuli, dokážeme tím, že existuje **alespoň jedna** nepravdivá valuace a tím pádem formule **není tautologií**. Nezapomínejme však na fakt, že jedna a táž proměnná může v jedné valuaci nabývat pouze jedné hodnoty (1/0).

Položíme-li si tedy otázku, zdali je možné najít takovou valuaci, určitě po chvíli přemýšlení dospějeme k závěru, že takovou valuaci nalézt lze. A to konkrétně p=0, q=0.

$$\neg 0 \supset (0 \vee 0)$$

Nalezli jsme tedy **valuaci**, pro kterou je formule **nepravdivá**, a tak jsme jednoduše dokázali, že formule **nemůže být tautologií**. Pokud by se nám tedy naopak nepovedlo takovou valuaci najít a vyzkoušeli jsme všechny možné kombinace, tak by daná formule tautologií byla.

2.2.2.3 Využití metody protipříkladu k dokazování kontradikcí

Obdobný postup metody protipříkladu můžeme také využít při dokazování **kontradikce**.

Postup bude skoro stejný, jako při dokazování tautologie, avšak tentokrát budeme hledat takovou **valuaci**, při které bude daná formule **pravdivá**. Pokud se nám takovou valuaci **podaří** nalézt, daná formule **není kontradikcí**.

Metoda protipříkladu **není** nijak implementována ve výsledném softwaru. Je zde ale zmíněna pro teoretickou ukázku další možné sémantické metody.

2.3 Normální formy formulí

Teď již víme, že ke každé formuli VL je jednoznačně přiřazena pravdivostní funkce, avšak k dané pravdivostní funkci může existovat mnoho formulí, které mají danou pravdivostní funkci za svou. Všechny tyto formule jsou samozřejmě navzájem ekvivalentní. Abychom tuto situaci zpřehlednili a zbavili se tak nejednoznačnosti, uvedeme si zde definici standardních (kanonických) tvarů formulí VL, kde *"každá třída navzájem ekvivalentních formulí bude reprezentována jednou formulí ve standardním tvaru",* jak uvádí prof. Duží ve své publikaci *Logika pro informatiky (a příbuzné obory)*: [2]

- "***Literál*** *– výroková proměnná nebo její negace.*"
- "***Elementární konjunkce (EK)*** *- konjunkce literálů*"
- "***Elementární disjunkce (ED)*** *- disjunkce literálů*"
- "***Úplná elementární konjunkce (UEK)*** *- UEK dané množiny výrokových proměnných je elementární konjunkce, ve které se každá proměnná z dané množiny vyskytuje právě jednou. Buďto prostě, nebo negovaná.*"
- "***Úplná elementární disjunkce (UED)*** *- UED dané množiny výrokových proměnných je elementární disjunkce, ve které ses každá proměnná z dané množiny vyskytuje právě jednou. Buďto prostě, nebo negovaná.*"
- "***Disjunktivní normální forma (DNF)*** *- DNF dané formule je formule ekvivalentní s danou formulí a mající tvar disjunkce elementárních konjunkcí.*"
- "***Konjunktivní normální forma (KNF)*** *- KNF dané formule je formule ekvivalentní s danou formulí a mající tvar konjunkce elementárních disjunkcí.*"
- "***Úplná disjunktivní normální forma (UDNF)*** *- UDNF dané formule je formule ekvivalentní s danou formulí a mající tvar disjunkce úplných elementárních konjunkcí.*"
- "***Úplná konjunktivní normální forma (UKNF)*** *- UDNF dané formule je formule ekvivalentní s danou formulí a mající tvar konjunkce úplných elementárních disjunkcí.*" [2]

V další kapitole této práce si uvedeme, jak převést formuli VL do některých z těchto forem a jak by správně tyto formy měly vypadat. Převod formulí do některých z těchto forem jsem totiž také implementoval do interaktivního softwaru. O způsobu implementace a realizace si však zmíníme až po kapitole, která bude věnována tomuto tématu.

3 Teorie syntaktických postupů pro ekvivalentní transformace, taktéž do KNF a DNF

Na konci minulé kapitoly jsme se dozvěděli o existenci různých normálních forem formulí VL. V této kapitole si nejdříve definujeme, co je to **ekvivalence** formulí, **ekvivalentní transformace**, a následně si objasníme převod formulí do těchto určitých forem, který se neobejde bez již zmíněných **ekvivalentních transformací**.

3.1 Ekvivalence a ekvivalentní transformace

Nejprve je na místě si uvést co to ekvivalence znamená. Formule jsou ekvivalentní, pokud mají právě stejné modely. Neboli ze sebe navzájem vyplývají. **Ekvivalence** tedy pro nás znamená vztah mezi libovolnými dvěma formulemi A, B, které nejsou od sebe odlišitelné z hlediska pravdivosti při žádné valuaci výrokových proměnných, které se vyskytují v obou proměnných pokládáme za sémanticky **ekvivalentní** [9].

Ekvivalentní transformace jsou tedy takové transformace, při kterých dochází k přetvoření původní formule, neboli k jistému přepisu na odlišnou formuli, která je však stále ekvivalentní k naší původní formuli. Velice užitečná věta pro lepší objasnění tohoto pojmu pochází z publikace *Logika v příkladech* a říká: *"Nechť CA je formule, která obsahuje jako samostatnou část formuli A. Nechť CB je formule, která vznikne tak, že v CA nahradíme výskyty formule A výskyty formule B. Pak platí-li |= A↔B (tj. jsou-li formule A a B logicky ekvivalentní) potom rovněž |= CA↔CB (neboli: rovněž formule CA a CB jsou logicky ekvivalentní)."* [10].

Ekvivalentních transformací existuje **nemalé množství**, avšak tady si uvedeme ty **nejznámější** a **nejvíce používané**.

Tabulka 1.11: *Přehled nejpoužívanějších ekvivalentních transformací*

Ekvivalence	**Název ekvivalentní úpravy**
$p \vee p \Leftrightarrow p$ $p \wedge p \Leftrightarrow p$	Zákon idempotentní
$p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$	Zákon komutativní
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	Zákon asociativní
$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$ $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	Zákon distributivní

$\neg (p \vee q) \Leftrightarrow \neg p \wedge \neg q$ $\neg (p \wedge q) \Leftrightarrow \neg p \vee \neg q$	De Morganův zákon
$p \vee (p \wedge q) \Leftrightarrow p$ $p \wedge (p \vee q) \Leftrightarrow p$	Zákon absorpce
$\neg (\neg p) \Leftrightarrow p$	Zákon dvojité negace / involuční zákon
$p \Leftrightarrow p$	Zákon totožnosti / identita
$p \vee \neg p \Leftrightarrow$ tautologie $p \wedge \neg p \Leftrightarrow$ kontradikce	Komplementárnost
$p \supset q \Leftrightarrow \neg p \vee q$	Úprava implikace
$p \equiv q \Leftrightarrow (\neg p \vee q) \wedge (p \vee \neg q)$	Úprava ekvivalence
$p \equiv q \Leftrightarrow (p \supset q) \wedge (q \supset p)$	Úprava ekvivalence pomocí implikace

Tyto **ekvivalentní transformace** můžeme využít k mnoha úpravám s formulemi VL. Chceme-li rozepsat danou formuli či naopak danou formuli zkrátit a napsat například její nejkratší možný tvar, využijeme právě ekvivalentní transformace.

Jako příklad efektivního využití ekvivalentních transformací si uvedeme zdánlivě nepříliš přehlednou formuli:

$$(\neg p \vee q) \wedge ((\neg r \vee s) \wedge (r \vee \neg s))$$

Na první pohled není hned jasné, která závorka se týká které operace, a orientace v této formuli tak není nejlepší. Formuli si však můžeme dosti efektivně přepsat a zkrátit. Pokud se podíváme pozorně na obsah v první závorce, zjistíme, že v závorce lze aplikovat ekvivalentní transformaci – a to úpravu **implikace**. Nebudeme však upravovat implikaci, ale naopak nahradíme výraz v závorce implikací. Je potřeba si uvědomit, že úpravy jsou ekvivalentní, tudíž platí oboustranně. Ve formuli tedy obsah první závorky nahradíme implikací následovně:

$$(p \supset q) \wedge ((\neg r \vee s) \wedge (r \vee \neg s))$$

Pokud se nyní podíváme na formuli, můžeme si všimnout, že obsah druhé závorky neboli pravého operandu první konjunkce ve formuli je ekvivalentní přepis ekvivalence. Můžeme si tedy obsah této závorky přepsat na ekvivalenci.

$$(p \supset q) \wedge (r \equiv s)$$

Nyní máme upravenou a zkrácenou formuli a orientace v této formuli je na první pohled snazší. Tuto formuli bychom samozřejmě mohli zpět převést do tvaru, který obsahoval pouze operace konjunkci a disjunkci tak, jak formule vypadala původně stejným postupem, jaký jsme provedli, avšak pozpátku.

V minulé kapitole jsme se dozvěděli o tom, že existuje hned několik **normálních forem formulí** VL. Tyto normální formy patří k **základům** učiva výrokové logiky a k tomu se také váže transformace neboli **způsob převodu** na danou normální formu. Právě k tomuto převodu nám mohou posloužit **ekvivalentní transformace**.

3.2 Disjunktivní a konjunktivní normální forma

Co je charakteristické pro tyto normální formy formulí je to, že si vystačí pouze se **třemi** výrokovými spojkami. Těmi jsou **negace** (¬), **konjunkce** (∧) a **disjunkce** (∨). V těchto formách se výrokové proměnné vyskytují se spojkou negace, nebo bez ní, tudíž jako literály [9]. Důležitým faktem je také to, že **každá** správně utvořená formule VL může být vyjádřena pouze s těmito třemi výrokovými spojkami a samozřejmě s výrokovými proměnnými.

Může existovat speciální účel, kdy je třeba vystihnout ty valuace, které vedou k pravdivé interpretaci dané formule. Právě pro tento účel je vhodná **disjunktivní normální forma** dané formule.

DNF – DISJUNKTIVNÍ NORMÁLNÍ FORMA – jak jsme se již dozvěděli je formule ekvivalentní s danou formulí a mající tvar **disjunkce elementárních konjunkcí**. **Definice 3 (DNF)**: *"Výroková formule je v disjunktivní normální formě, je-li disjunkcí podformulí (disjunktů), z nichž každá je konjunkcí konečně mnoha literálů jejích výrokových proměnných."* [3]. Jako příklad disjunktivní normální formy si můžeme uvést formuli:

(p ∧ q) ∨ (¬r ∧ s) ∨ (¬t ∧ ¬u)

Nebo další formuli:

(¬r ∧ s) ∨ p

KNF – KONJUNKTIVNÍ NORMÁLNÍ FORMA – spolu s disjunktivní normální formou také pochopitelně existuje KNF která má právě opačné uspořádání než DNF, tudíž je to formule ekvivalentní s danou formulí a mající tvar **konjunkce elementárních disjunkcí**. Před definicí KNF je nutné uvést definici pojmu klauzule. **Definice 4 (klauzule):** "*Řekněme, že klauzule ψ1 a ψ2 jsou v rezoluční relaci podle literálu θ, pokud jedna z nich obsahuje θ a druhá ¬θ. Mějme dvě klauzule ψ1 a ψ2 v rezoluční relaci podle θ, °e θ ∈ψ1 a ¬θ ∈ψ2. Rezolventa klauzulí ψ1 a ψ2 podle θ je nová klauzule označená: Rθ (ψ1, ψ2) = ψ1 \ {θ} ∪ψ2 \ {¬θ}."[3].*

Definice 5 (KNF): *"Výroková formule je v konjunktivní normální formě, resp.* v *klauzární formě, je-li konjunkcí podformulí (konjunktů), z nichž každá je disjunkcí konečně mnoha literálů jejích výrokových proměnných."* [3]. Konjunktivní normální formy jsou velmi užitečné pro různá počítačová využití, jako například pro databázové dotazování [1]. Jako příklad konjunktivní normální formy si můžeme uvést formuli:

(p ∨ q) ∧ (¬r ∨ s) ∧ (¬t ∨ ¬u)

Nebo další formuli:

$(\neg r \vee s) \wedge p$

Otázkou však zůstává, jakým způsobem převést libovolnou formuli právě do takovýchto podob a lze takto převést jakoukoli libovolnou formuli?

3.2.1 Transformace formule do DNF

K **převodu** libovolné formule VL do ekvivalentní disjunktivní normální formy se využívají výše zmíněné **ekvivalentní transformace**. **Každou** výrokovou formuli můžeme převést do ekvivalentní DNF [3]. Je definován tento konkrétní **postup**:

- *Eliminace výrokových spojek implikace (⊃) a ekvivalence (≡) pomocí funkčně úplné množiny {¬, ∧, ∨}.*
- Převod negací dovnitř závorek ve formuli za pomocí De Morganova pravidla.
- Eliminace dvojí negace.
- Užití asociativního a zákona pro konjunkci a disjunkci.
- Užití distributivního zákona (do potřebné podoby např: "(p ∧ q) ∨ (r ∧ s)") [3].

Tímto postupem jsme schopni vytvořit disjunktivní normální formule každé formule VL. Pro názornost můžeme postup této transformace uvést na následující formuli:

$p \wedge \neg (r \supset s)$

Prvním krokem, který musíme provést je **eliminace implikace**, která se nachází ve formuli. Provedeme tedy eliminaci a výsledná formule dostane podobu:

$p \wedge \neg (\neg r \vee s)$

Nyní je třeba přesunout všechny negace, které se vyskytují před závorkami dovnitř závorek. Tím docílíme negací dané závorky neboli použitím **De Morganova zákonu**. Po této transformaci bude formule mít následný tvar:

$p \wedge (\neg (\neg r) \wedge \neg s)$

V této fázi si můžeme povšimnout, že ve formuli je přítomná **dvojitá negace**. Naší povinností je tedy pro správnou tvorbu DNF tuto dvojitou negaci upravit. Získáme tak takovýto tvar formule:

$p \wedge (r \wedge \neg s)$

Nyní jsme se dostali do fáze, kdy je formule ve tvaru, který odpovídá disjunktivně normální formě. Stačí už jen odstranit zbytečnou závorku a dostaneme tak **výsledný tvar DNF**. Výsledná formule je však také v KNF:

$p \wedge r \wedge \neg s$

3.2.2 Transformace formule do KNF

S aktuální znalostí definic pojmů DNF a KNF je nutno podotknout, že pomocí záměny logických spojek **konjunkce** a **disjunkce** můžeme z jedné formy, ať už DNF nebo KNF získat tu druhou. Jak už víme, tak se tato vlastnost nazývá **dualita** [3].

Tak jako můžeme pro **každou** dobře utvořenou formuli VL vytvořit disjunktivní normální formu, tak můžeme taktéž ke každé takové formuli vytvořit **KNF**.

K **transformaci** formule VL do konjunktivní normální formy tak můžeme **využít totožný postup**, jaký jsme si uvedli u transformace formule do DNF. Změna však bude taková, že budeme cílit na výsledný tvar: "(p ∨ q) ∧ (r ∨ s)". Při těchto transformacích je tak nutno dbát zvýšené opatrnosti zejména při používání distributivního zákonu, abychom pak ve výsledku dostali správný tvar dané normální formy.

Pro názornost si ukážeme i postup při transformaci do KNF u této konkrétní formule:

$$p \vee \neg (r \supset s)$$

Opět zde máme přítomnou **implikaci**, a tak následuje nejprve její eliminace.

$$p \vee \neg (\neg r \vee s)$$

Následně musíme přesunout negaci dovnitř závorek pomocí **De Morganova zákonu**.

$$p \vee (\neg (\neg r) \wedge \neg s)$$

Nyní následuje úprava přítomné **dvojité negace**. Formule po úpravě bude mít podobu:

$$p \vee (r \wedge \neg s)$$

Tento tvar formule, který teď máme by odpovídal tvaru disjunktivní normální formy. My však chceme sestrojit **konjunktivní normální formu**, a tak je nutností ještě provést správnou úpravu využitím **distributivního zákona**. Po této transformaci se dostaneme k **výslednému tvaru KNF** této formule:

$$(p \vee r) \wedge (p \vee \neg s)$$

3.3 Úplná disjunktivní a úplná konjunktivní normální forma

V minulé podkapitole této práce jsme si představili dvě normální formy formulí VL, a to konkrétně DNF a KNF. Uvedli jsme si, že pro jakoukoli dobře utvořenou formuli VL existuje každá z těchto forem. Nyní si však představíme dvě další normální formy, které nemusí existovat ve všech případech formulí.

ÚDNF – ÚPLNÁ DISJUNKTIVNÍ NORMÁLNÍ FORMA – **Definice 6 (ÚDNF)**: *"Formule B je úplnou disjunktivní normální formou (ÚDNF) formule A právě tehdy, když B je ekvivalentní A, přičemž B je disjunkcí elementárních konjunkcí literálů z A, přičemž v každém jejím disjunktu se vyskytují všechny literály A právě jednou."* [1].

ÚKNF – ÚPLNÁ KONJUNKTIVNÍ NORMÁLNÍ FORMA – ÚKNF je naopak konjunkce úplných elementárních disjunkcí.

Na první pohled vypadají DNF a ÚDNF nebo KNF a ÚKNF velice podobně. Avšak postup, jakým se tyto formální normy vytváří je odlišný, a tak je odlišný i jejich obsah.

3.3.1 Transformace formule do ÚDNF

Způsobů, jak transformovat formuli VL do úplně disjunktivní normální formy, je více. ÚDNF můžeme sestrojit pomocí **ekvivalentních úprav**, nebo také za pomocí využití **pravdivostní tabulky**, a právě tento způsob si představíme. Tento způsob je opět vhodný pro situace, kdy formule **neobsahuje** příliš velké množství výrokových proměnných, aby výsledná pravdivostní tabulka tak nebyla příliš velká. Pro následnost si tento způsob předvedeme na formuli:
"¬p ∧ (q ⊃ r)"

Tabulka 1.12: *Pravdivostní tabulka formule: "¬p ∧ (q ⊃ r)" - ÚDNF*

p	**q**	**r**	**¬p ∧ (q ⊃ r)**
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Když se podíváme na tuto pravdivostní tabulku výše uvedené formule, tak si jistě povšimneme, že tato formule má **3 modely** neboli 3 valuace vedou k pravdivému ohodnocení celé formule. **Postup** pro **utvoření ÚDNF** naší formule je následující:

- Vyčteme všechny pravdivé valuace v tabulce, kterým poté přiřadíme elementární konjunkci všech výrokových proměnných.
- V případě, kdy v pravdivé valuaci má některá z proměnných přiřazenou hodnotu 0, je nutno před danou proměnnou do výsledné elementární konjunkce umístit znak negace.
- Tato získaná ohodnocení – pro nás elementární konjunkce – poté seřadíme do formule a spojíme tyto jednotlivé elementární konjunkce pomocí spojky disjunkce.

V naší tabulce tedy nalezneme pravdivé valuace a zapíšeme si je jako elementární konjunkce následovně:

¬p ∧ ¬q ∧ ¬r

¬p ∧ ¬q ∧ r

¬p ∧ q ∧ r

Nesmíme však zapomenout na zmíněnou negaci těch proměnných, které mají v dané valuaci přiřazenou hodnotu 0. Nyní nám stačí už jen spojit tyto tři elementární konjunkce pomocí spojky disjunkce. Výsledkem je už výsledná ÚDNF:

$$(\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r)$$

3.3.2 Transformace formule do ÚKNF

Pro převod formule VL do úplné konjunktivní normální formy můžeme opět využít **ekvivalentních úprav**, nebo opět využít metodu pomocí **pravdivostní tabulky**. Metoda převodu formule VL do ÚKNF je obdobná, jako u převodu do ÚDNF. Zde si opět uvedeme konkrétní postup:

- Vyčteme všechny nepravdivé valuace v tabulce, kterým poté přiřadíme elementární disjunkci všech výrokových proměnných.
- V případě, kdy v nepravdivé valuaci má některá z proměnných přiřazenou hodnotu 1, je nutno před danou proměnnou do výsledné elementární disjunkce umístit znak negace.
- Tato získaná ohodnocení – pro nás elementární disjunkce – poté seřadíme do formule a spojíme tyto jednotlivé elementární disjunkce pomocí spojky konjunkce.

Pro názornost si opět uvedeme příklad převodu formule: "$\neg p \wedge (q \supset r)$" do ÚKNF. Nejprve si tedy sestrojíme **pravdivostní tabulku** dané formule.

Tabulka 1.13: *Pravdivostní tabulka formule: "$\neg p \wedge (q \supset r)$" - ÚKNF*

p	**q**	**r**	**$\neg p \wedge (q \supset r)$**
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Vypíšeme si tedy všechny valuace, při kterých nabývá formule **nepravdivé** hodnoty jako **elementární disjunkci**. Nezapomeňme k proměnným, které mají hodnotu **1**, přiřadit **negaci**.

$p \vee \neg q \vee r$

$\neg p \vee q \vee r$

$\neg p \vee q \vee \neg r$

$\neg p \vee \neg q \vee r$

$\neg p \vee \neg q \vee \neg r$

Následně už je potřeba tyto elementární disjunkce spojit pomocí logické operace **konjunkce** a dostaneme výslednou **ÚKNF**:

$(p \vee \neg q \vee r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$

Nyní již víme, jak vypadají formy **ÚDNF** a **ÚKNF**, a také, jak je sestrojit. Proto je teď důležité zmínit fakt, že ke každé formuli VL, která **není kontradikcí**, lze nalézt ekvivalentní formuli ve tvaru **ÚDNF**, a naopak pro každou formuli VL, která **není tautologií**, lze nalézt ekvivalentní formuli ve tvaru **ÚKNF**. Zkrátka k formuli, která je kontradikcí, nenajdeme ÚDNF a k formuli, která je tautologií, nejsme schopni nalézt ÚKNF.

3.4 Implementace generování DNF a KNF do softwaru

Pro velkou využitelnost těchto normálních forem jsem se rozhodl zaimplementovat jejich generaci do **interaktivního softwaru**. Funkčnost je taková, že pokud uživatel zadá správně utvořenou formuli VL, software začne postupně generovat zvolenou formu. Postupná generace probíhá formou interaktivního kvízu. Uživateli je vždy zobrazena aktuální formule a název ekvivalentní transformace, která se v dalším kroku provede. Uživatel je poté dotázán, jak bude formule po konkrétní ekvivalentní transformaci vypadat a uživatel má na výběr ze 4 možností, z čehož je vždy pouze **jedna správná**. Při zadání správné odpovědi se odpověď označí **zelenou** barvou. Naopak při špatné odpovědi se odpověď uživatele označí **červenou** barvou, avšak ukáže se mu také správná odpověď. Poté se celý systém opakuje pro další ekvivalentní úpravu, která následuje. Až uživatel tak dojde do poslední fáze, kdy už zbyde pouze formule ve správném DNF nebo KNF tvaru, program uživatele informuje, že se jedná o konečnou a správnou formu.

Způsob transformace zadané formule, který byl pro tuto funkci využit, jde ruku v ruce s teorií o tvorbě DNF a KNF, která byla zmíněna výše. Pro vstupní formuli tedy software nejprve provede **úpravu pořadí** v dané formuli. To znamená, že dle potřeby přidá nutné závorky pro správnou orientaci v dané formuli. Poté software provede **úpravu ekvivalence**, pokud se v dané formuli nějaká nachází, a eliminuje ji pomocí dvou nově vzniklých implikací. Následuje **úprava přítomných implikací**, po které už ve formuli zbydou pouze operátory: $\neg$, $\wedge$ a $\vee$. Jako další v řadě se přesunou negace dovnitř závorek dle principu **De Morganova zákona** a následně se provede detekce a úprava možných **dvojitých negací**. V neposlední řadě se pak provádí **distributivní úprava** a na samotný konec je provedena úprava za pomocí principu **idempotentního zákonu** na celkové zjednodušení výsledné formule.

Výsledný tvar DNF a KNF tak nemusí být vždy v tom nejkratším tvaru dané formy, ale jedná se o správně utvořené formy DNF a KNF.

Využitím této funkce v interaktivním softwaru ve výuce by tak mohlo dojít ke zefektivnění a lepší organizaci výuky a ke snazšímu porozumění této problematiky.

3.5 Implementace generace ÚDNF a ÚKNF do softwaru

V softwaru jsou také implementovány neméně důležité normální formy **ÚDNF** a **ÚKNF**. Při tvorbě funkcí na jejich generaci jsem opět postupoval dle již popsaných teoretických postupů, a to konkrétně dle postupu transformace do ÚDNF/ÚKNF pomocí **pravdivostní tabulky**. Ve fázi, kdy jsem tvořil tuto funkci, jsem již měl hotovou funkci pro generaci pravdivostní tabulky formule VL, a tak zbývalo jen tuto funkci správně využít.

Základní stavební jednotkou při generaci těchto normálních forem byla tedy nejprve vygenerovaná **pravdivostní tabulka** dané formule. Následně jen stačilo projít pole, ve kterém byla uložena všechna výsledná ohodnocení formule. Pro tvorbu **ÚDNF** bylo potřeba, jak již víme, najít všechna **pravdivá** ohodnocení. Když bylo konkrétní ohodnocení pravdivé, zapsaly se stranou jako jednotlivé **elementární konjunkce** všechny výrokové proměnné. Bylo potřeba také kontrolovat, jakou hodnotu mají proměnné v dané valuaci. Pokud proměnné měly přiřazenou **pravdivou** hodnotu neboli 1, bylo potřeba před tyto proměnné umístit znak **negace**. Nakonec už byly tyto elementární konjunkce umístěny do závorek a spojeny pomocí logické operace **disjunkce**. Algoritmus pro tvorbu **ÚKNF** byl samozřejmě totožný jen s výjimkou detekce **nepravdivých** ohodnocení místo pravdivých a negací proměnných s **nepravdivou** hodnotou namísto pravdivé.

Software počítá s tím, že v případě **tautologie** neexistuje **ÚKNF**, a naopak v případě **kontradikce** není schopen nalézt **ÚDNF**.

Funkce ale funguje tak, že nejprve pro uživatele vygeneruje **pravdivostní tabulku** pro zadanou formuli. Následně si funkce vygeneruje formy **ÚKNF** A **ÚDNF**, avšak tyto formy nejsou uživateli zobrazeny. Je to právě naopak. Uživatel musí za pomocí vygenerované pravdivostní tabulky **sestrojit** výslednou formu ÚDNF či ÚKNF sám, a poté si nechá výsledek programem zkontrolovat. Program poté dá uživateli na vědomí, zda sestrojil danou formu správně, či nikoliv.

4 Implementace a otestování interaktivního softwaru

4.1 Implementace jednotlivých funkcí v softwaru

Způsob **implementace** jednotlivých částí byl již stručně popsán u jednotlivých již uvedených částí. Zde si rozebereme a popíšeme postupnou implementaci jednotlivých částí celého interaktivního softwaru více dopodrobna a uvedeme si, co dané funkce využívají z hlediska **programové části**.

Na počátku samotného nápadu pro tvorbu interaktivního softwaru, který mohl sloužit jako didaktická pomůcka ve výuce výrokové logiky, bylo třeba dobře znát a orientovat se v základech výrokové logiky, které popisuje tato práce.

Interaktivní software je určen pro **mobilní zařízení** s operačním systémem Android 8 nebo vyšší, a tak bylo pro tvorbu tohoto softwaru použito vývojové prostředí **Android Studio**, se kterým jsem se již setkal v minulosti ve výuce bakalářského studia. Jako programovací jazyk jsem zvolil **Javu**. Program je tedy cílen na operační systém s minimální verzí **8.0 nebo vyšší**.

Jako základní stavební kámen v této mobilní aplikaci bylo na začátku vytvořeno grafické **uživatelské rozhraní** v podobě klasického mobilního kalkulátoru. To znamená, že bylo potřeba udělat přehledná tlačítka pro všechny znaky, pomocí kterých uživatel bude následně moci sestavovat dobře utvořené formule VL. Toto rozhraní bylo **základem** pro všechny následně vytvořené funkce, jelikož každá z těchto funkcí vyžaduje jako vstup právě dobře utvořenou formuli VL. Se zadáváním vstupu v tomto rozhraní jde samozřejmě ruku v ruce kontrola správnosti vstupu, která nedovolí uživateli zadat nesprávně utvořenou formuli, a dává mu také na vědomí, na které pozici udělal uživatel danou chybu.

Po sestavení základu pro možnost zadání správného vstupu následovalo vytvoření první použitelné funkce. Tou byla funkce úlohy s pravdivostní tabulkou. Víme, že pravdivostní tabulka bývá hojně využívána při další práci s formulemi výrokové logiky, a tak bylo důležité sestavit generaci pravdivostní tabulky jako první možnou použitelnou funkci. Při tvorbě pravdivostní tabulky byly přesně následovány teoretické postupy pro její tvorbu. Hlavní využití při tvorbě této funkce měly **arrayListy** neboli **dynamické pole**, které fungují jako seznamy postavené nad polem. Využití arrayListu bylo pro naši problematiku ideální řešení, jelikož poskytoval pole s **dynamickým přístupem**, který je potřeba právě při tvorbě pravdivostní tabulky. ArrayListy byly tedy využity pro zápis jednotlivých znaků unikátních proměnných, jednotlivých číselných ohodnocení (konkrétních valuací) a také pro zápis celkových ohodnocení pro uživatelem zadanou formuli. Při úspěšném a konečném sestavení těchto dynamických polí už následoval jen výsledný výpis všech použitých unikátních **výrokových proměnných** a následně všech **valuací** a jejich výsledných **vyhodnocení** pro zadanou vstupní formuli. Výsledná funkce tak zobrazuje uživateli formuli, kterou zadal, a k ní příslušnou pravdivostní tabulku. Pro vyhodnocení dané valuace byl, jak je již popsáno v této práci, využit způsob převedení dané formule s daným ohodnocením z datového typu string do datového typu boolean.

Klasický jazyk **Java** jako takový tento převod ze stringu do booleanu neumožňuje, pokud jsou ve výrazu použity logické operace a výraz tak není jen pravdivostní hodnota true nebo false, a tak byla pro tento převod využita knihovna Rhino, která poskytovala funkci na převod naší aktuální formule s danou valuací z datového typu string do datového typu boolean. Byl kladen důraz i na správné zobrazení příliš dlouhých pravdivostních tabulek, a to tím, že výsledné zobrazení funkce podporuje rolování neboli scrolling v dané pravdivostní tabulce. Výsledná pravdivostní tabulka je tedy vyobrazena uživateli, a ten pak může vyplněním políček odpovídat na dotazy, zdali je zadaná formule dle její pravdivostní tabulky splnitelná, tautologií, kontradikcí a jaký má počet modelů. Funkce je následně schopna snadno zkontrolovat tyto odpovědi, jelikož má díky vygenerované pravdivostní tabulce rychlý přehled, jaká ohodnocení se v tabulce nachází.

Nyní víme, že za pomocí pravdivostní tabulky můžeme jednoduše sestavit normální formy **ÚDNF** a **ÚKNF** pro danou formuli VL. Zmínili jsme si však, že při příliš velkých pravdivostních tabulkách může být hledání daných pravdivých či nepravdivých ohodnocení pro následné sestavení těchto forem velice nepřehledné a také zdlouhavé.

To však v tomto případě platí pouze pro lidské oko, ale ne pro dnešní **výkonná mobilní zařízení**, která si s tímhle úkonem hravě poradí ve velice nízkém čase. Zařízení tak stačí jen procházet arrayList s výslednými ohodnoceními a kontrolovat, jestli jsou **pravdivá** či **nepravdivá**, a následně si z ostatních arrayListů vzít **výrokové proměnné**. Dle aktuální valuace se také výrokovým proměnným přiřazuje znak **negace**, či nikoliv. Na konci této funkce tak byla potřeba pouze pospojovat daná **ohodnocení** příslušnou logickou operací. Výsledná funkce funguje tedy na principu vygenerování ÚDNF nebo ÚKNF a následně vyzve uživatele, aby jednu z těchto forem sestrojil sám díky přítomné pravdivostní tabulce. Program je poté schopen zkontrolovat správnost odpovědi uživatele.

Další důležitou funkcí byla generace normálních forem **DNF** a **KNF**. Pro realizaci této funkce byla opět následována teoretická pravidla, kdy výsledné normální formy sestavíme pomocí ekvivalentních úprav. Hlavním bodem bylo tedy sestrojení algoritmů pro dané ekvivalentní úpravy, které jsou nezbytné ke správné tvorbě DNF a KNF. V tomto ohledu byly v Javě využity regulární výrazy, a to konkrétně třídy pattern a matcher. Za pomocí třídy matcher můžeme totiž ověřit, zdali nějaký textový řetězec – v našem případě daná formule či podformule obsahuje či splňuje kritéria, která jsou daná regulárním výrazem, což je v tomto případě vždy nějaká šablona (text) – objekt pattern. Tento způsob tak byl využit pro kontrolu, jestli daná formule či podformule obsahuje takový výraz, který by bylo možné ekvivalentně upravit. Kontroluje se tedy přítomnost ekvivalence, implikace, dále jestli je možné provést úpravu dle De Morganova zákona či úpravu dvojité negace a v neposlední řadě, jestli je možno provést distributivní úpravu, a nakonec, jestli je možno provést idempotentní úpravu. Způsob pomocí využití tříd pattern a matcher byl tedy využit pro každou z těchto ekvivalentních úprav. Následná realizace ekvivalentních úprav pak byla provedena opět za využití kolekce arrayList, který za celou dobu běhu těchto ekvivalentních úprav představoval celkovou strukturu aktuální formule. V tomto arrayListu, který představoval celkovou a poté i výslednou strukturu dané formule, byla využita reprezentace daných podformulí či závorek pomocí čísel.

Dané **číslo** tak vždy odkazuje na **index daného objektu v arrayListu**, který obsahuje danou **podformuli**. Pomocí tohoto systému pak byla snazší manipulace s danými podformulemi, například jejich negace či tvorba nových podformulí, která je využita hlavně v distributivní úpravě. Za pomocí tohoto principu se mi tak povedlo vytvořit fungující systém, který dokáže pracovat s kteroukoli dobře utvořenou formulí VL a dokáže detekovat, jestli je možné provést všechny ekvivalentní úpravy nezbytné pro správnou tvorbu DNF a KNF, a následně je také realizovat. Program tak postupně krok po kroku upravuje zadanou formuli na danou **DNF** či **KNF** formu a uživatel je před každým krokem dotázán formou **kvízu**, jak bude po úpravě formule vypadat. Kroky s danými ekvivalentními transformacemi se provádějí vždy, i když není nutnost potřeba některé transformace. Na otázku, jak bude tedy vypadat formule po transformaci, která není třeba provést by měl uživatel odpovědět stejným tvarem formule, jaký je před zmíněnou úpravou.

Následující funkce pouze využívá již realizovaných ekvivalentních úprav. Tato funkce totiž funguje tak, že dá uživateli možnost zadat správně utvořenou formuli VL. Následně si může vybrat z **následujících ekvivalentních úprav**:

- Úprava dvojité negace
- Úprava dle De Morganova zákonu
- Úprava dle distributivního zákonu pro ∧, ∨
- Úprava dle distributivního zákonu pro ∨, ∧
- Eliminace implikace
- Eliminace ekvivalence pomocí spojek ∧, ∨
- Úprava dle idempotentního zákonu

Výběr těchto ekvivalentních úprav byl zvolen, protože tyto úpravy patří mezi jedny z nejčastěji používaných a nejvíce základních. Zároveň však chybí úprava dle **asociativního** a **komutativního** zákonu. Tyto úpravy zde nejsou zahrnuty, jelikož upravují jen pořadí daných výrokových proměnných. Při velkém množství proměnných by se také nabízelo příliš mnoho možností, jak tyto úpravy provést, a nelze tak jasně definovat pevný algoritmus, který by dokázal tyto úpravy vždy provádět. Uživatel si tak následně může vybrat vždy **jednu** z nabízených ekvivalentních úprav. Je-li to možné, program provede danou ekvivalentní úpravu nad zadanou formulí a vypíše uživateli **výsledek** – **správně upravenou ekvivalentní formuli**. Je nutno podotknout, že program vždy provede zvolenou transformaci ve **všech možných případech**. To znamená, že pokud formule obsahuje například vetší množství přítomných implikací, a jako ekvivalentní transformace je zvolena eliminace implikace, tak program upraví všechny přítomné implikace v dané formuli. Uživatel tak získá možnost volby jedné z těchto základních ekvivalentních transformací a následné rychlé vyobrazení již transformované ekvivalentní formule.

Poslední funkcí, kterou si uživatel může v tomto interaktivním softwaru zvolit, je **kontrola správného provedení ekvivalentních úprav**. Funkce funguje tak, že uživatel nejprve vybere jednu ekvivalentní transformaci ze seznamu podporovaných transformací.

Následně je uživatel vyzván, aby zadal první formuli a druhou, k ní ekvivalentní formuli po zvolené transformaci. Program tedy vyhodnotí správnost použití dané transformace a informuje uživatele o výsledku.

Podporované ekvivalentní transformace jsou:

- Úprava dvojité negace
- Úprava dle De Morganova zákonu
- Úprava dle distributivního zákonu
- Eliminace implikace
- Eliminace ekvivalence pomocí spojek ∧, ∨
- Úprava dle idempotentního zákonu
- Úprava dle asociativního nebo komutativního zákonu

Důležité je však dodržet správný postup pro správnou funkčnost této funkce. Je třeba tedy počítat, že funkce je schopna kontrolovat pouze ekvivalentní transformace. Funkce také funguje tak, že je vždy třeba provést **pouze jednu ekvivalentní transformaci**, a také jí provést ve **všech případech**. To znamená, že pokud provádíme například úpravu ekvivalence, je třeba upravit všechny přítomné ekvivalence ve formuli (toto se netýká pouze úpravy dle idempotentního zákonu). Při vypsání výsledné upravené formule je tedy nutno dodržet stejnou strukturu formule, jakou má zadaná první formule, a provést pouze jednu ekvivalentní úpravu ve všech případech. Funkce tak dokáže provést kontrolu nad správností provedení dané ekvivalentní transformace a dát také uživateli na vědomí, zda se vůbec o ekvivalentní úpravu jedná. Pokud jsou zadané formule totožné, nebo nebyla provedena žádná z podporovaných úprav, program to dá uživateli na vědomí.

Každý **vstup** pro každou funkci je také **ověřován** a je nutností, aby uživatel zadal **správně utvořenou formuli** VL. Pokud uživatel ve vstupu udělá chybu a formule tak není správně utvořená, program jasně vypíše, že se ve vstupu nachází chyba a na jaké pozici se nachází.

Po kompletaci všech těchto funkcí, které výsledný interaktivní software nabízí, bylo potřeba uspořádat výběr těchto funkcí do přehledného **menu**. Vytvořil jsem tak uvítací menu, které uživatele uvítá a vyzve ho, aby si zvolil jednu z nabízených funkcí.

Software také nabízí možnost ukončovat dané funkce a přecházet zpět do menu či opětovně zadávat nové formule pro každou z naimplementovaných funkcí. Každá z funkcí včetně menu je ve vývojovém prostředí Android Studio reprezentována jako **aktivita**. Aktivity jsou ošetřeny řádným **ukončením** po jejich opuštění anebo jejich **restartováním** při opětovném využití.

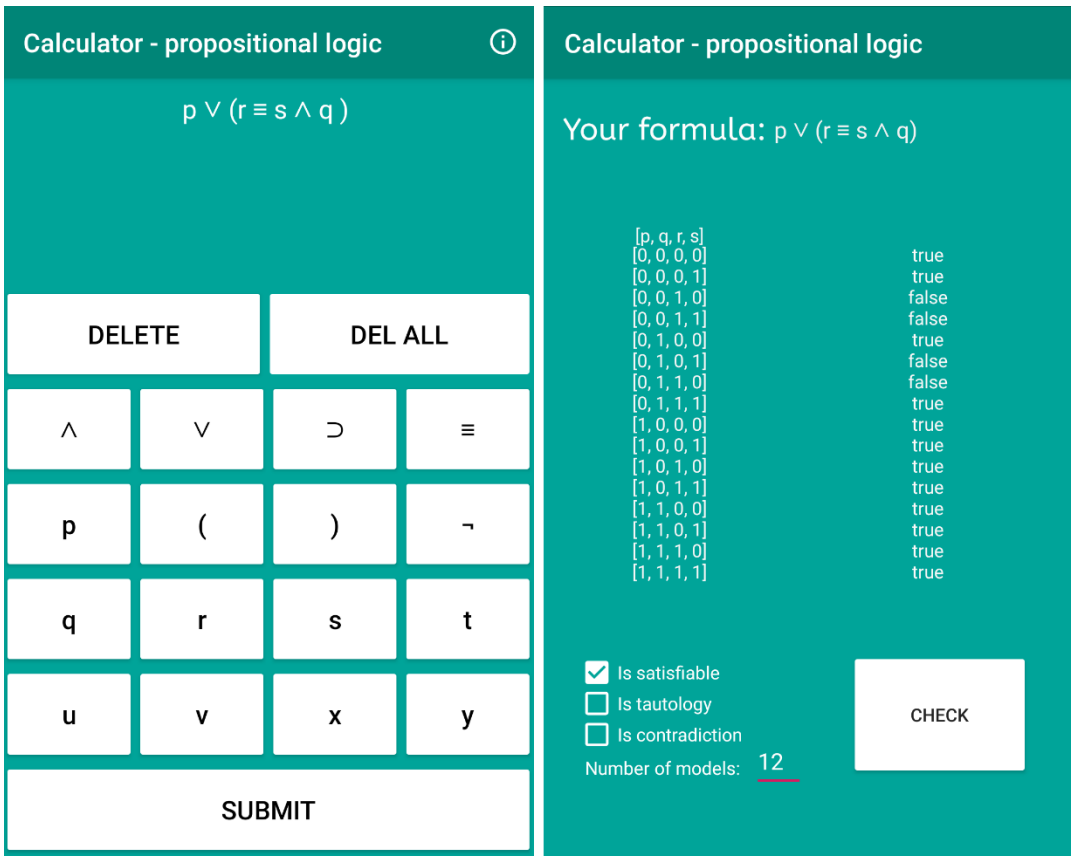
Software také nabízí tlačítko pro **informaci** o dané funkci – jejím používání a funkčnosti. Toto tlačítko je implementováno ve všech funkcích tohoto softwaru.

Výsledný interaktivní software se tak podařilo naimplementovat s využitím teoretických základů a postupů, které jsou uváděny v této práci.

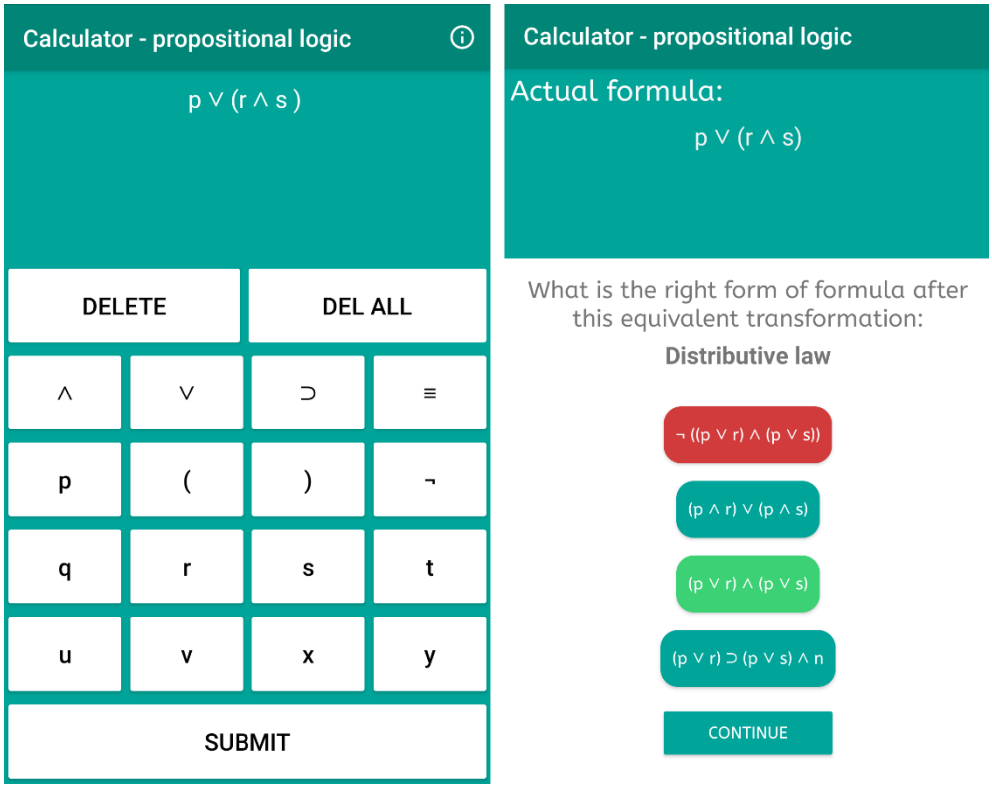
Software tak může sloužit jako interaktivní didaktická pomůcka do výuky výrokové logiky, jelikož nabízí širokou škálu využití, a je schopen pomoci s organizací a efektivností výuky této problematiky, a tudíž vést k jejímu lepšímu porozumění.

4.2 Otestování jednotlivých funkcí v softwaru

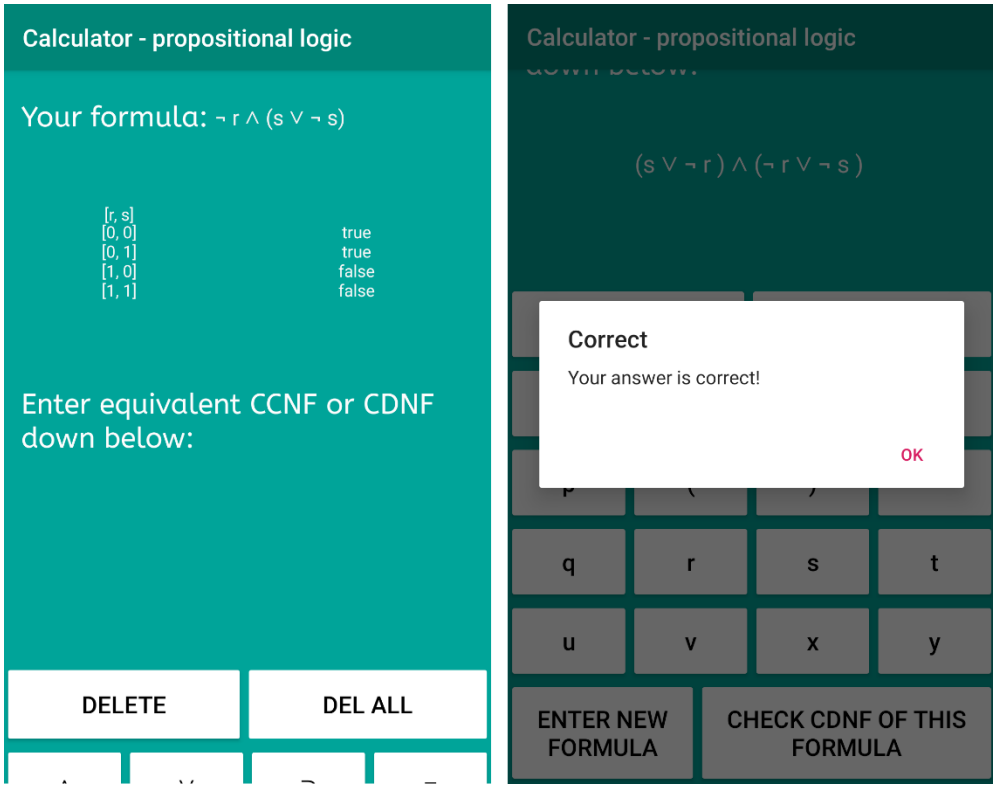
V této podkapitole je demonstrováno otestování výsledného softwaru za pomocí obrázků z aplikace. Postupně bude vyobrazena každá funkce s dvěma příklady a jejich výsledky.



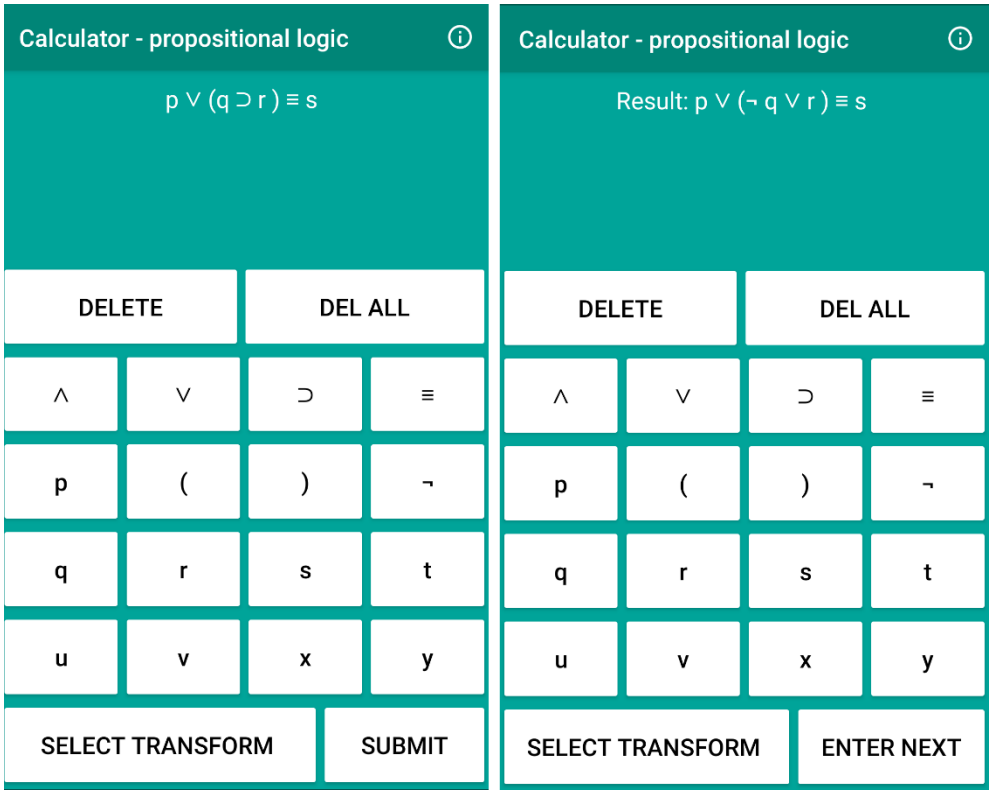
Obrázek 1.1: *Otestování funkce úlohy s pravdivostní tabulkou. 1*



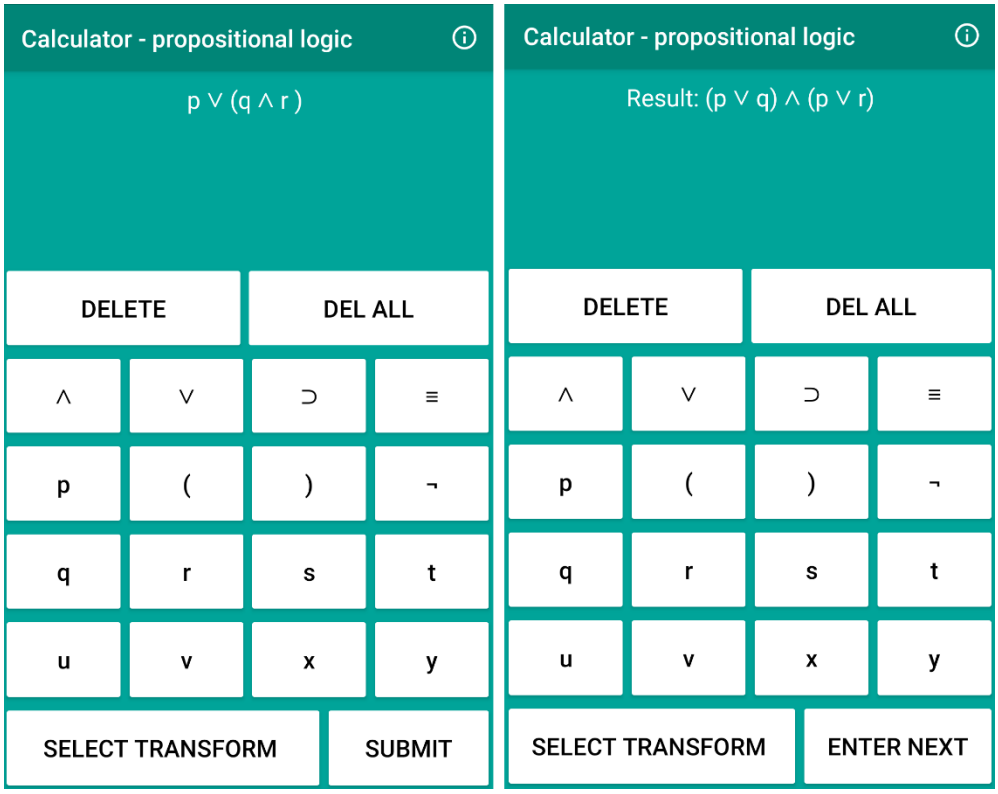
Obrázek 1.2: *Otestování interaktivní generace KNF. 2*



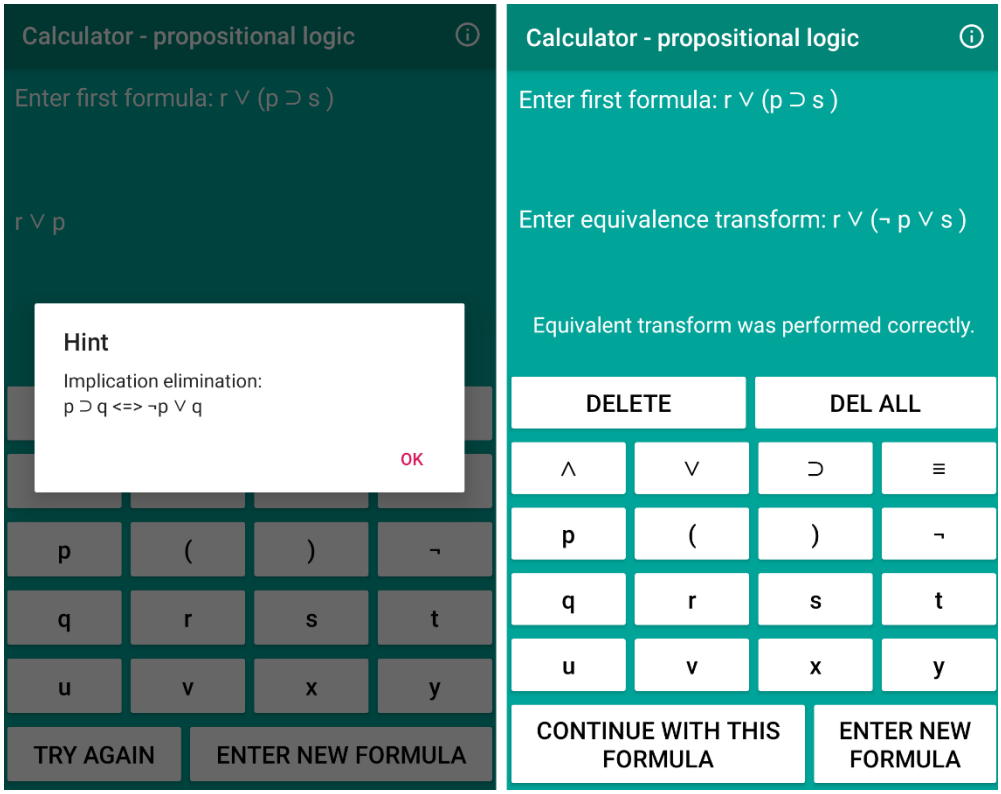
Obrázek 1.3: *Otestování generace ÚKNF dle pravdivostní tabulky. 3*



Obrázek 1.4: *Otestování výběru ekvivalentní transformace v softwaru – eliminace implikace. 4*

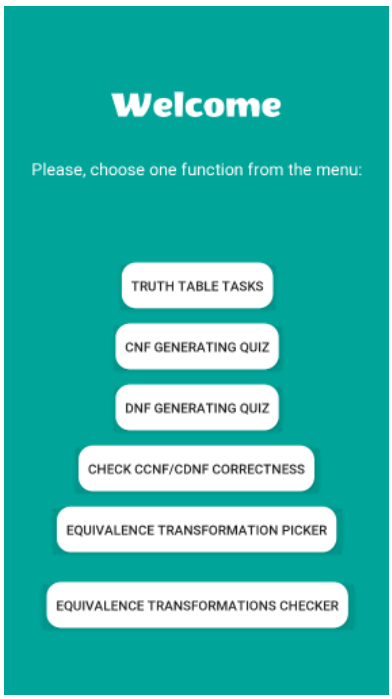


Obrázek 1.5: *Otestování výběru ekvivalentní transformace v softwaru – distributivní úprava. 5*



Obrázek 1.6: *Otestování kontroly správnosti ekvivalentních úprav – eliminace implikace. 6*

Na závěr této podkapitoly je zde přidán obrázek uvítacího menu aplikace, kde si uživatel vybírá, jakou funkci chce použít.



Obrázek 1.7: *Uvítací menu v softwaru. 7*

Závěr

Cílem této bakalářské práce bylo probrat základy výrokové logiky, její sémantiky a taktéž ekvivalentních úprav s pozorností na normální formy formulí a následně naimplementovat a také otestovat interaktivní software, který by mohl sloužit jako pomůcka při výuce úvodu do výrokové logiky. Software by měl být určen pro mobilní zařízení se systémem Android 8.0, nebo vyšší. Stručně jsme si objasnili teorii základů výrokové logiky a ukázali si teoretické postupy při ekvivalentních transformacích a také postupy při transformacích do nejznámějších a nejpoužívanějších normálních forem formulí výrokové logiky.

Zjistili jsme tak, že teoretické postupy pro ekvivalentní úpravy a transformace do normálních forem jsou dostatečným a jasným postupem pro realizaci těchto úprav a transformací v programátorském prostředí. Výsledkem této práce je interaktivní software, který dokáže přijmout správně utvořené formule výrokové logiky a provádět s nimi operace, které patří k základům výrokové logiky. Software tak může být přínosem ve výuce tohoto učiva a může výrazně přispět k lepšímu porozumění této problematiky. Výsledná aplikace je také optimalizována pro většinu druhů obrazovek samotných zařízení včetně tabletů a také pro správné zobrazení při otáčení zobrazení různých zařízení.

Nabízí se také další vývoj tohoto interaktivního softwaru, a to například o doplnění mnohých dalších operací, které se ve výrokové logice využívají či rozšíření tohoto softwaru na více zařízení nebo realizace softwaru v podobě interaktivních webové stránky. Je možná také návaznost s pracemi zabírajících se ostatními logikami, jako je například predikátová logika, a tedy rozšíření softwaru o další nedílnou součást výuky na mnoha vysokých školách.

Použitá literatura

[1] RACLAVSKÝ, Jiří. Úvod do logiky: klasická výroková logika. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-7790-4.

[2] DUŽÍ, Marie. Logika pro informatiky (a příbuzné obory): učební text. Ostrava: VŠB-TU Ostrava, 2012. ISBN 978-80-248-2662-2.

[3] LUKASOVÁ, Alena. Formální logika v umělé inteligenci. Brno: Computer Press, 2003. ISBN 80-251-0023-5.

[4] PEREGRIN, Jaroslav. Logika a logiky: systém klasické výrokové logiky, jeho rozšíření a alternativy. Praha: Academia, 2004. ISBN 80-200-1187-0.

[5] ŠVEJDAR, Vítězslav. Logika: neúplnost, složitost a nutnost. Praha: Academia, 2002. ISBN 80-200-1005-x.

[6] ŠTĚPÁN, Jan. Formální logika. 2., přeprac. vyd. Olomouc: Fin, 1995. ISBN 80-7182-004-0.

[7] SVOBODA, Vladimír a Jaroslav PEREGRIN. Od jazyka k logice: filozofický úvod do moderní logiky. Praha: Academia, 2009. Galileo. ISBN 978-80-200-1740-6.

[8] GRAND, Mark. Java: referenční příručka jazyka. Praha: Computer Press, c1998. ISBN 80-7226-071-5.

[9] LUKASOVÁ, Alena. Logické základy umělé inteligence I: [studijní materiály pro distanční kurz ...]. Ostrava: Ostravská univerzita, 2003. Systém celoživotního vzdělávání Moravskoslezska. ISBN 80-7042-846-5.

[10] HROMEK, Petr. Logika v příkladech. Olomouc: Univerzita Palackého, 2002. ISBN 80-244-0578-4.

Seznam příloh

Příloha v IS EDISON:

- navodNaInstalaci.pdf - Návod pro instalaci aplikace (.apk souboru) na mobilní zařízení.
- plCalculator.apk - Aplikace pro mobilní zařízení s operačním systémem android 8.0 nebo vyšší
- plCalculator.zip - Zdrojový kód softwaru (projet tvořen v Android Studio).

Příloha A: *Návod na instalaci aplikace*

Návod na instalaci softwaru (soubor .apk)

Pro instalaci softwaru je nutno mít operační systém Android verze 8.0 nebo vyšší.

1. Přesunout plCalculator.apk na mobilní zařízení.

2. V nastavení zařízení povolit „Instalaci z neznámých zdrojů“.

3. Na zařízení nalézt a otevřít a dále nainstalovat soubor plCalculator.apk.

4. Aplikace se poté objeví mezi nainstalovanými aplikacemi s názvem „PL – Interactive tool“.

Pro správnou funkčnost daných funkcí je třeba vždy zadat dobře utvořenou formuli výrokové logiky s těmi znaky, které aplikaci nabízí. Pro bližší nápovědu obsahuje každá funkce tlačítko „i“.